CARTOGRAPHER: a tool for string diagrammatic reasoning

Paweł Sobociński
University of Southampton, UK

Paul Wilson
University of Southampton, UK and University College London, UK

Fabio Zanasi
University College London, UK

Abstract

We introduce CARTOGRAPHER , a tool for editing and rewriting string diagrams of symmetric monoidal categories. Our approach is principled: the layout exploits the isomorphism between string diagrams and monogamous cospans of hypergraphs; the implementation of rewriting is based on the soundness and completeness of convex double-pushout rewriting for string diagram rewriting.

2012 ACM Subject Classification Visual Languages

Keywords and phrases tool, string diagram, symmetric monoidal category, graphical reasoning

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.

1 Introduction

String diagrammatic theories are increasingly important in computer science. They have been recently been used in a number of applications, including enabling the simplification of quantum circuits using the ZX-calculus [10], compositional descriptions of models of concurrency such as Petri Nets [18, 6], compositional accounts of signal flow graphs in control theory [7, 11, 1] and Bayesian reasoning [8, 14, 13]. These examples, as well as many others, work with the language of *symmetric monoidal categories* (SMCs). This paper addresses the need for tool support for *symmetric monoidal theories* - graphical rewriting systems of SMCs.

CARTOGRAPHER is a graphical editor and proof assistant for symmetric monoidal theories. It provides a graphical string diagram editor to construct morphisms, and a prover in which rewrite rules can be specified and executed. Further, CARTOGRAPHER has a firm theoretical foundation, its rewriting backend based on recent work in the area [5, 3, 20, 4]. The goal of this paper is to motivate CARTOGRAPHER , explain the basic features of the backend and the front end, and describe some of the technical challenges that were solved in creating it. The tool and its user guide are available on the CARTOGRAPHER website at `http://cartographer.id/`.

Our motivating example is the rewriting system in Figure 1. The intended semantic interpretation is that of binary circuits, where each wire carries an n bit number for some fixed n. Green nodes with two outputs copy numbers, those with no outputs discard their input, while red nodes perform addition modulo 2^n.

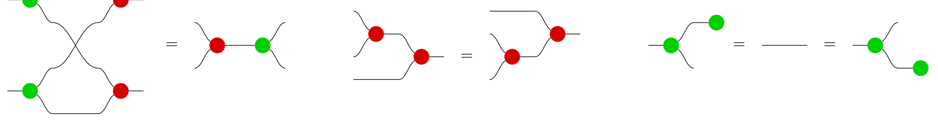


Figure 1 Example rules for binary circuits with copying ( ), adding ( ), and discarding ( ).

As well as the rules in Figure 1, this rewriting system implicitly uses three *generators*; atomic sub-diagrams, each with some number of inputs and outputs. These are the *copy* ( ), *add* ( ), and *discard* ( ) operations. The laws of symmetric monoidal categories permit moving generators around up to an isotopy made precise in [15, 19]. For our purposes, it suffices to say informally that generators can be slid along wires, and moved around on the page, but not rotated. By way of example, consider the equivalent diagrams in Figure 2.

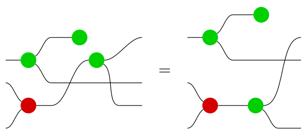


Figure 2 Example of string diagrams considered equal under the laws of SMCs

CARTOGRAPHER allows reasoning modulo the laws of symmetric monoidal categories. The user can deform morphisms up to the SMC laws without making proofs unsound, and the prover does not require (e.g. when matching the l.h.s. of a rule) the user to explicitly use the laws of symmetric monoidal categories. Put another way, the user should not have to "untangle" the wires of the diagram before applying a rule of some theory.

To put this into context, compare CARTOGRAPHER to two "competing" tools: Quantomatic [17] and Globular [2] (or its more recent descendant, `homotopy.io`). In a sense, CARTOGRAPHER sits between them: providing a more general setting than Quantomatic, while at the same time being more focussed than Globular.

software	generality	geometric intuition
Quantomatic	compact-closed	generators can implicitly be moved and wires bent back
Cartographer	symmetric monoidal	generators can implicitly be moved
Globular	higher categories	no implicit deformations permitted

Quantomatic deals with (less general) *compact closed* categories, in which not only may generators be moved, but wires may be "bent backwards". In terms of our circuit analogy, this would mean feedback, e.g. as used in a simple latch. CARTOGRAPHER allows such feedback, but as an explicit compact closed structure in the theory at hand, not implicitly assumed to exist by the underlying tool. On the other hand, Globular is much more general, aiming to support diagrammatic reasoning in higher categories. While this allows more freedom, when working with SMCs it comes at the cost of having to explicitly use SMC laws in proofs, e.g. using the functoriality of the monoidal product to slide two generators past each other.

Contributions

The contribution of CARTOGRAPHER is twofold. First, in the back end we implement an algorithm for matching and rewriting modulo the laws of SMC based on the adequacy result of [5]. The algorithm works with a data structure for *Open Hypergraphs*, which we introduce in this paper. Second, in the front end, we use an algorithm for the layout of these directed acyclic open hypergraphs which behaves well under rewriting and deformation of diagrams.

2 Directed Acyclic Open Hypergraphs

The main problem of implementing rewriting modulo symmetric monoidal structure is in finding a data structure in which equivalent terms have a single representation. For example, the two equivalent diagrams of Figure 2 should have the same underlying representation. Our approach is principled, because it uses the isomorphism between equivalent terms and cospans of hypergraphs found in [5]. Starting from this result, we propose an alternative but equivalent representation which is more convenient to work with.

We begin with an overview of the open hypergraphs of [5], and the CARTOGRAPHER data structure— illustrated in Figure 3 along with the corresponding string diagram. Beginning with the central (open) hypergraph, hyperedges are denoted , and represent the generators of the string diagram. There are two kinds of nodes, denoted •. Firstly (ordered) boundary nodes, connected to a single wire (input or output, but not both) Secondly, internal nodes, having exactly one input and one output wire, thus satisfying monogamicity [5].

In contrast, the hypergraphs of CARTOGRAPHER are closed, and so nodes are rendered simply as wires, each with exactly one input and one output connection. Boundary nodes are replaced by adding special generators to the signature of the hypergraph, s (boundary source) and t (boundary target). Nodes are then uniquely identified by the two "ports" they connect— a *port* being a specific position on the boundary of a hyperedge.

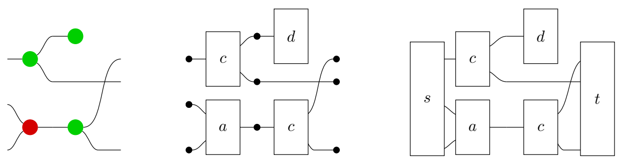


Figure 3 From left to right: a string diagram, its open hypergraph representation with signature $\Sigma = \{a, c, d\}$, and the equivalent closed hypergraph with signature $\Sigma' = \Sigma \cup \{s, t\}$

▶ **Definition 1.** A $k \to m$ CARTOGRAPHER *hypergraph* (Σ, E, W) consists of:

- the *signature* Σ, which can be thought of as the set of *types* of hyperedges. Each has arity $ar : \Sigma \to \mathbb{N} \times \mathbb{N}$, giving the number of inputs and outputs. We require that the Σ contains *boundary* generators σ, τ, with $ar(\sigma) = (0, k)$ and $ar(\tau) = (m, 0)$;
- the set of *hyperedges* E, with a function $typ : E \to \Sigma$ that assigns types to hyperedges. Moreover, there are *boundary* hyperedges $\{s, t\} \subseteq E$ s.t. $typ^{-1}(\sigma) = \{s\}$, $typ^{-1}(\tau) = \{t\}$;
- the set of *wires* W. Given a hyperedge $e \in E$, if $dim(type(e)) = (p, q)$ then we say e has p input ports, denoted $e^1, e^2, \ldots, e^q$, and q output ports denoted $e_1, e_2, \ldots, e_q$. A *wire* $w \in W$ is an ordered pair (e_i, f^j) of a *source port* e_i and a *target port* f^j, denoting a directed connection from the i^{th} output of e to the j^{th} input of f.

3 Visualising and Editing Open Hypergraphs

In contrast to Quantomatic [17] which uses a force-directed layout, and Globular [2] which has a fixed style for morphism layout, we use a *layered graph drawing* algorithm similar to that of Dot [12]. Our reasons for choosing layered graph drawing are as follows. Firstly, it was an aesthetic choice to represent string diagrams similarly to how they appear in the literature. Secondly, string diagrams drawn with the layered discipline retain a closer link with the underlying algebraic description of morphisms, since the term can by easily be read off the string diagram in the form of a composition-of-monoidal-products. Thirdly, in contrast to *force-directed* approaches, the elements of a layered graph layout do not move around on the page, which is problematic from a user-experience perspective, because they are harder for the user to click. Additionally, force-directed layouts can change significantly after a rewrite rule is applied, with little control over the resulting diagram. This can be confusing for the user, because the string diagram may look very different. Finally, using layered hypergraphs offers a simple and intuitive way to enforce acyclicity: users may only connect generators if the target appears to the right of the source.

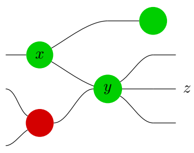
Interactive Layered Graph Drawing

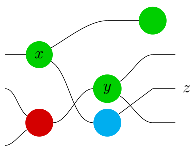
We briefly summarise the interactive layered graph drawing approach of CARTOGRAPHER . By "interactive", we mean to distinguish CARTOGRAPHER 's layout algorithm from other layered graph drawing approaches—such as Dot's—in which a static graph is given as input, and positions of nodes and edges are returned. CARTOGRAPHER allows for the *incremental construction* of hypergraphs, meaning that users begin with a blank canvas, and add generators and connections one-by-one. We call it a layered graph drawing approach because it uses two key ideas from those approaches: the user of *layers*, and of *pseudonodes*.

▶ **Definition 2.** Given a CARTOGRAPHER hypergraph (Σ, E, W) and $e \neq e' \in E$, there is a directed path from e to e' if there exists a sequence $(e_1, \dots, e_n)$ where $e_i \in E$, $e_1 = e$, $e_n = e'$ and for each e_i, e_{i+1} there exist j_1, j_2 such that $((e_i)_{j_1}, (e_{i+1})^{j_2}) \in W$. A *layering* is a function $L : E \to \mathbb{N}$ such that:

(i) if there is a directed path from e to e' then $L(e) < L(e')$;
(ii) for every non-boundary hyperedge $e \in E$, $L(s) < L(e) < L(t)$.

The layering L essentially serves as the "x coordinate" of each hyperedge. The second idea from layered graph layout is the use of *pseudonodes*, which are conceptually related to the edge-points of Dixon and Kissinger's Open Graphs [9], but used here only for layout purposes: they prevent wires from crossing generators. For a concrete example of why this is desirable, consider Figure 4. In the left-hand diagram, the wire from x to z passes through y and it is not clear whether x is connected to y and y to z, or if x is directly connected to z. Inserting pseudonodes into the graph clears up the ambiguity.





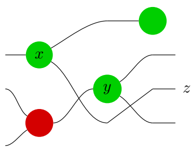


Figure 4 Left, a diagram with only generators (rendered ● and ●), center, the same diagram after inserting pseudonodes (rendered ●), and right, the diagram as it appears with pseudonodes hidden.

The Layout Algorithm

We briefly outline the layout algorithm used in CARTOGRAPHER . Because the algorithm is interactive, it takes the form of a *layout state*, and a number of *actions* that the user can take. We model these actions as functions of the layout state.

The layout state is a tuple (H, G) of a hypergraph H as in definition 1, and an *integer grid* G, which keeps track of the positions of generators and pseudonodes as two dimensional vectors. Users can perform two actions on the layout state:

1. Placing a generator at a specific position on an integer grid
2. Moving a generator from one position to another
3. Connecting a source port to a target port

Moving and placing a generator is straightforward: if a generator e is moved or placed such that it would overlap with another generator f, then f is moved down within the same layer to make space. However, when connecting ports we must ensure that the hypergraph H remains acyclic. This is enforced using the following constraints:

- If generators e, f have layers such that $L(e) \leq L(f)$, then outputs of f may not be connected to inputs of e.
- If a generator f is reachable from e, then f may not be moved such that $L(f) \leq L(e)$.

These constraints ensure that layering respects the properties of Definition 2, preserving acyclicity. Finally, for every operation, the set of required pseudonodes is maintained, along with their positions in G. In particular, this means updates for any operation which changes connectivity, or modifies the number of layers between two generators.

4 Matching, Convexity, Rewriting

As well as an interactive string diagram editor, CARTOGRAPHER enables diagrammatic reasoning. A derivation consists of a series of rewrites, using a set of rules specified by the user. A *rule* consists of two CARTOGRAPHER hypergraphs, the lhs and the rhs, with identical boundaries. Rewriting is implemented by double-pushout rewriting of hypergraphs, with soundness and completeness guaranteed by [5, Theorem 5.6].

Applying a rule to a string diagram consists of three steps: finding a match for the lhs a rule, checking for convexity, and applying the rewrite rule. A *match* is an hypergraph embedding (an injective, homomorphic mapping of hyperedges and nodes) of open hypergraphs, with one subtlety: the boundary ports of the pattern match can map to non-boundary ports in the target. CARTOGRAPHER builds matches incrementally by using the backtracking logic library *logict* [16]. Roughly speaking, wires and generators are added to the working match until either there are no more unmatched wires or generators, or a contradiction is reached, in which case the search backtracks. Candidate matches are then checked for *convexity* [5], which is needed for a rewrite to be valid modulo the laws of SMCs. Roughly speaking, all directed paths that start and end in a matched region must remain within the match. Once a convex match has been identified, the internal hyperedges of the matched region are removed and replaced with the right hand side of the rewrite rule.

5 Conclusions and Future Work

CARTOGRAPHER is still in early stages of development. We are working on

- improving the layout algorithms by adapting heuristics from other tools that work with layered graphs;
- more advanced features for diagrammatic reasoning, including support for structured proofs (using e.g. user-generated Lemmas) and adapting other user-friendly features originally developed for theorem provers and proof assistants;
- higher level specification features, such as support for bang-boxes, recursive definitions, and proof strategies;
- better decoupling between the rewriting back end and the layout front end, enabling extensions such as rewriting modulo compact closed structure.

References

1 John Baez and Jason Erbele. Categories in control. *Theory and Applications of Categories*, 30:836–881, 2015.

2 Krzysztof Bar, Aleks Kissinger, and Jamie Vicary. Globular: an online proof assistant for higher-dimensional rewriting. In *Leibniz International Proceedings in Informatics*, volume 52, pages 34:1–34:11, 2016. ncatlab.org/nlab/show/Globular.

3 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Soboci'nski, and Fabio Zanasi. Confluence of graph rewriting with interfaces. In *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, pages 141–169, 2017. URL: `https://doi.org/10.1007/978-3-662-54434-1_6`, `doi:10.1007/978-3-662-54434-1\_6`.

4 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Soboci'nski, and Fabio Zanasi. Rewriting with frobenius. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 165–174, 2018. URL: `https://doi.org/10.1145/3209108.3209137`, `doi:10.1145/3209108.3209137`.

5 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. Rewriting modulo symmetric monoidal structure. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science - LICS '16*, pages 710–719, New York, NY, USA, 2016. ACM Press. URL: `http://dl.acm.org/citation.cfm?doid=2933575.2935316`, `doi:10.1145/2933575.2935316`.

6 Filippo Bonchi, Joshua Holland, Robin Piedeleu, Pawel Soboci'nski, and Fabio Zanasi. Diagrammatic algebra: from linear to concurrent systems. *PACMPL*, 3(POPL):25:1–25:28, 2019. URL: `https://dl.acm.org/citation.cfm?id=3290338`.

7 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. The calculus of signal flow diagrams I: linear relations on streams. *Inf. Comput.*, 252:2–29, 2017.

8 Benjamin Cabrera, Tobias Heindel, Reiko Heckel, and Barbara König. Updating probabilistic knowledge on condition/event nets using bayesian networks. In *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, pages 27:1–27:17, 2018. URL: `https://doi.org/10.4230/LIPIcs.CONCUR.2018.27`, `doi:10.4230/LIPIcs.CONCUR.2018.27`.

9 Lucas Dixon and Aleks Kissinger. Open-graphs and monoidal theories. *Mathematical Structures in Computer Science*, 23(2):308–359, 2013.

10 Andrew Fagan and Ross Duncan. Optimising Clifford Circuits with Quantomatic. *Electronic Proceedings in Theoretical Computer Science*, 287:85–105, January 2019. URL: `http://arxiv.org/abs/1901.10114v1`, `doi:10.4204/EPTCS.287.5`.

11 Brendan Fong, Paolo Rapisarda, and Paweł Sobociński. A categorical approach to open and interconnected dynamical systems. In *Thirty-first annual ACM/IEEE symposium on Logic and Computer Science (LiCS 2016)*, pages 495–504, 2016. `doi:10.1145/2933575.2934556`.

12 E.R. Gansner, E. Koutsofios, S.C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, March 1993. URL: `http://ieeexplore.ieee.org/document/221135/`, `doi:10.1109/32.221135`.

13 Bart Jacobs, Aleks Kissinger, and Fabio Zanasi. Causal inference by string diagram surgery. 2019. URL: `http://arxiv.org/abs/1811.08338`.

14 Bart Jacobs and Fabio Zanasi. The logical essentials of bayesian reasoning. In Joost-Peter Katoen Gilles Barthe and Alexandra Silva, editors, *Probabilistic Programming*. Cambridge University Press, Cambridge, 2019. URL: `http://arxiv.org/abs/1804.01193`.

15 Andre Joyal and Ross Street. The geometry of tensor calculus, I. *Adv. Math.*, 88:55–112, 1991.

16 Oleg Kiselyov, Chung-chieh Shan, Daniel P Friedman, and Amr Sabry. Backtracking, Interleaving, and Terminating Monad Transformers. page 12.

17 Aleks Kissinger and Vladimir Zamdzhiev. Quantomatic: A Proof Assistant for Diagrammatic Reasoning. *arXiv:1503.01034 [cs, math]*, 9195:326–336, 2015. arXiv: 1503.01034. URL: http://arxiv.org/abs/1503.01034, doi:10.1007/978-3-319-21401-6_22.

18 José Meseguer and Ugo Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105–155, October 1990. URL: https://linkinghub.elsevier.com/retrieve/pii/0890540190900138, doi:10.1016/0890-5401(90)90013-8.

19 Peter Selinger. A survey of graphical languages for monoidal categories. *arXiv:0908.3347 [math]*, 813:289–355, 2010. arXiv: 0908.3347. URL: http://arxiv.org/abs/0908.3347, doi:10.1007/978-3-642-12821-9_4.

20 Fabio Zanasi. Rewriting in free hypegraph categories. In *Proceedings Third Workshop on Graphs as Models, GaM@ETAPS 2017, Uppsala, Sweden, 23rd April 2017.*, pages 16–30, 2017. URL: https://doi.org/10.4204/EPTCS.263.2, doi:10.4204/EPTCS.263.2.